Java Database Connectivity (JDBC): A Comprehensive Guide

Dr. Ratnesh Prasad Srivastava, Associate Professor, CSIT, GGU

Introduction

JDBC (Java Database Connectivity) is a Java API that enables Java applications to interact with databases. It provides a standard interface for connecting to relational databases, executing SQL queries, and processing results.

JDBC Architecture Components

- **JDBC API**: Provides interfaces and classes for database operations
- **JDBC Driver Manager**: Manages database drivers
- **JDBC Drivers**: Vendor-specific implementations that connect to databases
- **Database**: The actual database system (MySQL, Oracle, PostgreSQL, etc.)

JDBC Architecture Overview

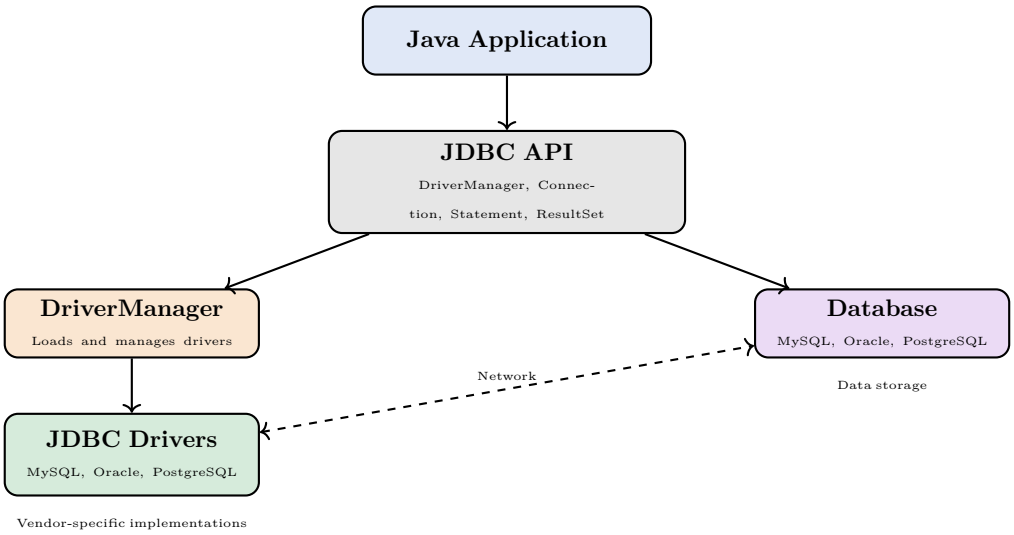

Figure 1: JDBC Architecture Overview

JDBC Core Interfaces and Classes

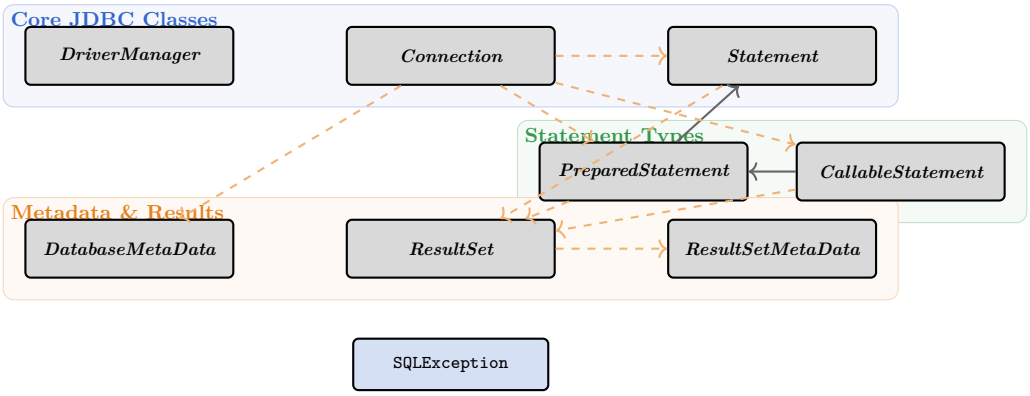

Figure 2: JDBC Core Interfaces and Relationships

Understanding JDBC Components

- **DriverManager**: Manages database drivers and establishes connections
- **Connection**: Represents a session with the database
- **Statement**: Executes static SQL queries
- **PreparedStatement**: Precompiled SQL statements with parameters
- **CallableStatement**: Executes stored procedures
- **ResultSet**: Represents result data from queries
- **SQLException**: Handles database access errors

JDBC Driver Types

Type	Name	Description
1	JDBC-ODBC Bridge	Translates JDBC to ODBC; deprecated, not recommended
2	Native-API Driver	Uses database client libraries; requires native code
3	Network-Protocol Driver	Middleware server translates calls; platform independent
4	Thin Driver	Pure Java driver; converts JDBC directly to database protocol

Table 1: JDBC Driver Types

JDBC Connection Process

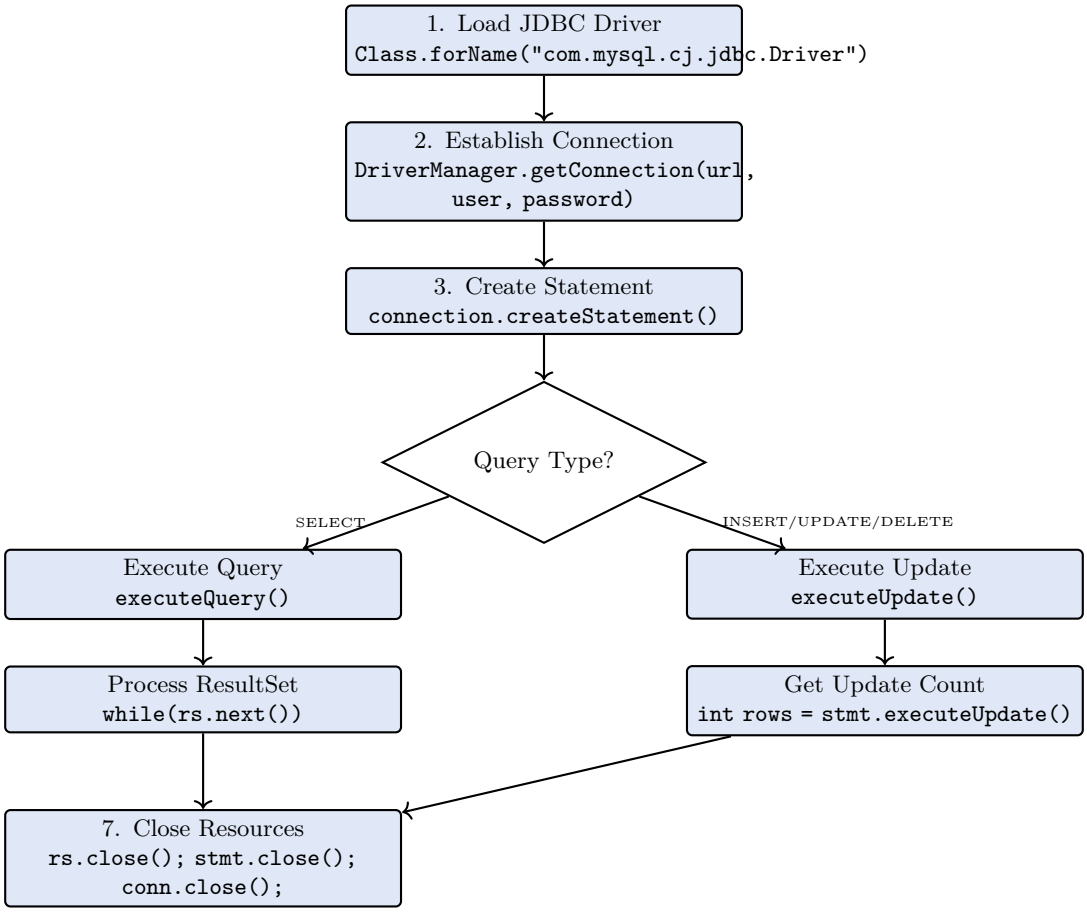

Figure 3: JDBC Connection and Query Execution Process

JDBC Statement Types Comparison

Feature	Statement	PreparedStatement	CallableStatement
Use Case	Static SQL queries	Parameterized queries	Stored procedures
Performance	Compiled each time	Precompiled, faster for repeated use	Optimized stored procedure
Security	Vulnerable to SQL injection	Prevents SQL injection	Secure parameter passing
Parameters	No parameters	Uses ? placeholders	IN/OUT parameters
Syntax	`stmt.executeQuery(sql)`	`pstmt.setString(1, value)`	`cstmt.registerOutParameter(...)`

Table 2: Statement Types Comparison

ResultSet Types and Concurrency

ResultSet Type	Description
TYPE_FORWARD_ONLY	Cursor can only move forward (default)
TYPE_SCROLL_INSENSITIVE	Scrollable; not sensitive to database changes
TYPE_SCROLL_SENSITIVE	Scrollable; sensitive to database changes
CONCUR_READ_ONLY	Cannot update ResultSet (default)
CONCUR_UPDATABLE	Can update ResultSet and reflect changes

Table 3: ResultSet Types and Concurrency Modes

The Demo Program

The following program demonstrates JDBC operations with MySQL database:

```
import java.sql.*;
import java.util.*;

public class AllJdbcDemo {

    // Database connection parameters
    private static final String URL = "jdbc:mysql://localhost:3306/testdb";
    private static final String USER = "root";
    private static final String PASSWORD = "password";

    public static void main(String[] args) {

        // ========== 1. LOAD DRIVER AND CONNECT ==========
        System.out.println("=== JDBC Connection Demo ===\n");

        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD)) {
            System.out.println (" Connected to database successfully!");

            // ========== 2. CREATE TABLE ==========
            createTable(conn);

            // ========== 3. INSERT DATA - Statement ==========
            insertUsingStatement(conn);

            // ========== 4. INSERT DATA - PreparedStatement ==========
            insertUsingPreparedStatement(conn);

            // ========== 5. QUERY DATA - Statement ==========
            queryUsingStatement(conn);

            // ========== 6. QUERY DATA - PreparedStatement with Parameter ==========
            queryUsingPreparedStatement(conn);

            // ========== 7. UPDATE DATA ==========
            updateData(conn);

            // ========== 8. DELETE DATA ==========
            deleteData(conn);

            // ========== 9. BATCH PROCESSING ==========
            batchProcessing(conn);

            // ========== 10. TRANSACTION MANAGEMENT ==========
            transactionDemo(conn);

            // ========== 11. METADATA ==========
            metadataDemo(conn);

            // ========== 12. SCROLLABLE RESULT SET ==========
            scrollableResultSetDemo(conn);

            // ========== 13. DATABASE METADATA ==========
            databaseMetadataDemo(conn);

        } catch (SQLException e) {
            System.err.println("Database error: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private static void createTable(Connection conn) throws SQLException {
        String sql = "CREATE TABLE IF NOT EXISTS employees (" +
                     "id INT PRIMARY KEY AUTO_INCREMENT, " +
                     "name VARCHAR(100) NOT NULL, " +
                     "department VARCHAR(50), " +
                     "salary DECIMAL(10,2), " +
                     "joining_date DATE)";

        try (Statement stmt = conn.createStatement()) {
            stmt.execute(sql);
```

```
            System.out.println(" Table 'employees' created/verified");
        }
    }

    private static void insertUsingStatement(Connection conn) throws SQLException {
        String sql = "INSERT INTO employees (name, department, salary, joining_date) " +
                     "VALUES ('John Doe', 'IT', 75000, '2023-01-15')";

        try (Statement stmt = conn.createStatement()) {
            int rows = stmt.executeUpdate(sql);
            System.out.println(" Statement: Inserted " + rows + " row(s)");
        }
    }

    private static void insertUsingPreparedStatement(Connection conn) throws SQLException {
        String sql = "INSERT INTO employees (name, department, salary, joining_date) " +
                     "VALUES (?, ?, ?, ?)";

        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, "Jane Smith");
            pstmt.setString(2, "HR");
            pstmt.setDouble(3, 65000);
            pstmt.setDate(4, Date.valueOf("2023-02-20"));
            int rows = pstmt.executeUpdate();

            pstmt.setString(1, "Bob Johnson");
            pstmt.setString(2, "IT");
            pstmt.setDouble(3, 82000);
            pstmt.setDate(4, Date.valueOf("2023-03-10"));
            rows += pstmt.executeUpdate();

            System.out.println(" PreparedStatement: Inserted " + rows + " row(s)");
        }
    }

    private static void queryUsingStatement(Connection conn) throws SQLException {
        String sql = "SELECT * FROM employees WHERE department = 'IT'";

        try (Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            System.out.println("\n--- IT Department Employees (Statement) ---");
            while (rs.next()) {
                System.out.printf("ID: %d, Name: %s, Salary: %.2f%n",
                    rs.getInt("id"),
                    rs.getString("name"),
                    rs.getDouble("salary"));
            }
        }
    }

    private static void queryUsingPreparedStatement(Connection conn) throws SQLException {
        String sql = "SELECT * FROM employees WHERE salary > ?";

        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setDouble(1, 70000);

            try (ResultSet rs = pstmt.executeQuery()) {
                System.out.println("\n--- Employees with Salary > 70000 ---");
                while (rs.next()) {
                    System.out.printf("Name: %s, Department: %s, Salary: %.2f%n",
                        rs.getString("name"),
                        rs.getString("department"),
                        rs.getDouble("salary"));
                }
            }
        }
    }

    private static void updateData(Connection conn) throws SQLException {
        String sql = "UPDATE employees SET salary = salary * 1.1 WHERE department = ?";

        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
```

```
            pstmt.setString(1, "IT");
            int rows = pstmt.executeUpdate();
            System.out.println(" Updated " + rows + " employee(s) in IT department (+10% salary)");
        }
    }

    private static void deleteData(Connection conn) throws SQLException {
        String sql = "DELETE FROM employees WHERE name = ?";

        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, "John Doe");
            int rows = pstmt.executeUpdate();
            System.out.println(" Deleted " + rows + " employee(s)");
        }
    }

    private static void batchProcessing(Connection conn) throws SQLException {
        String sql = "INSERT INTO employees (name, department, salary, joining_date) VALUES (?, ?, ?, ?)";

        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            conn.setAutoCommit(false);

            pstmt.setString(1, "Alice Brown");
            pstmt.setString(2, "Finance");
            pstmt.setDouble(3, 72000);
            pstmt.setDate(4, Date.valueOf("2023-04-01"));
            pstmt.addBatch();

            pstmt.setString(1, "Charlie Wilson");
            pstmt.setString(2, "Marketing");
            pstmt.setDouble(3, 68000);
            pstmt.setDate(4, Date.valueOf("2023-04-15"));
            pstmt.addBatch();

            int[] results = pstmt.executeBatch();
            conn.commit();
            conn.setAutoCommit(true);

            System.out.println(" Batch processed " + results.length + " record(s)");
        }
    }

    private static void transactionDemo(Connection conn) throws SQLException {
        System.out.println("\n--- Transaction Demo ---");
        conn.setAutoCommit(false);

        try {
            String sql1 = "UPDATE employees SET salary = salary + 5000 WHERE department = 'IT'";
            String sql2 = "UPDATE employees SET salary = salary + 3000 WHERE department = 'HR'";

            try (Statement stmt = conn.createStatement()) {
                stmt.executeUpdate(sql1);
                stmt.executeUpdate(sql2);
            }

            conn.commit();
            System.out.println(" Transaction committed successfully");

        } catch (SQLException e) {
            conn.rollback();
            System.err.println(" Transaction rolled back: " + e.getMessage());
        } finally {
            conn.setAutoCommit(true);
        }
    }

    private static void metadataDemo(Connection conn) throws SQLException {
        String sql = "SELECT * FROM employees WHERE id = 1";

```

```
        try (Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            ResultSetMetaData rsmd = rs.getMetaData();
            System.out.println("\n--- ResultSet Metadata ---");
            System.out.println("Column count: " + rsmd.getColumnCount());

            for (int i = 1; i <= rsmd.getColumnCount(); i++) {
                System.out.printf("Column %d: %s (%s)%n",
                    i,
                    rsmd.getColumnName(i),
                    rsmd.getColumnTypeName(i));
            }
        }
    }

    private static void scrollableResultSetDemo(Connection conn) throws SQLException {
        Statement stmt = conn.createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_READ_ONLY);

        ResultSet rs = stmt.executeQuery("SELECT id, name, salary FROM employees ORDER BY id")
    ;

        System.out.println("\n--- Scrollable ResultSet ---");

        if (rs.last()) {
            System.out.println("Last record: " + rs.getString("name"));
            System.out.println("Total records: " + rs.getRow());
        }

        if (rs.first()) {
            System.out.println("First record: " + rs.getString("name"));
        }

        rs.absolute(2);
        System.out.println("Record at position 2: " + rs.getString("name"));

        rs.previous();
        System.out.println("Previous record: " + rs.getString("name"));

        rs.close();
        stmt.close();
    }

    private static void databaseMetadataDemo(Connection conn) throws SQLException {
        DatabaseMetaData dbmd = conn.getMetaData();

        System.out.println("\n--- Database Metadata ---");
        System.out.println("Database: " + dbmd.getDatabaseProductName());
        System.out.println("Version: " + dbmd.getDatabaseProductVersion());
        System.out.println("Driver: " + dbmd.getDriverName());
        System.out.println("JDBC Version: " + dbmd.getJDBCMajorVersion() + "." +
                           dbmd.getJDBCMinorVersion());
        System.out.println("Supports transactions: " + dbmd.supportsTransactions());
        System.out.println("Supports batch updates: " + dbmd.supportsBatchUpdates());

        // List tables
        ResultSet tables = dbmd.getTables(null, null, "%", new String[]{"TABLE"});
        System.out.println("\nTables in database:");
        while (tables.next()) {
            System.out.println("  - " + tables.getString("TABLE_NAME"));
        }
        tables.close();
    }
}
```

Listing 1: Comprehensive JDBC Demonstration

Code Clarifications

1. **Driver Loading**: `Class.forName()` loads the JDBC driver (automatic in JDBC 4.0+)
2. **Connection**: `DriverManager.getConnection()` establishes database connection
3. **Statement**: Used for static SQL queries without parameters
4. **PreparedStatement**: Precompiled SQL with parameters; prevents SQL injection
5. **ResultSet**: Represents query results; iterates with `next()`
6. **Batch Processing**: `addBatch()`/`executeBatch()` for bulk operations
7. **Transactions**: `setAutoCommit(false)` enables manual transaction control
8. **Metadata**: `ResultSetMetaData` and `DatabaseMetaData` provide database info
9. **Scrollable ResultSet**: Allows navigation in both directions
10. **Resource Management**: Try-with-resources automatically closes connections

JDBC Best Practices

- **Use PreparedStatement**: Prevents SQL injection and improves performance
- **Manage Connections**: Use connection pools for production applications
- **Handle Transactions**: Commit/rollback appropriately for data integrity
- **Close Resources**: Always close ResultSet, Statement, and Connection
- **Use try-with-resources**: Automatic resource management in Java 7+
- **Handle Exceptions**: Catch SQLException and provide meaningful error messages
- **Batch Operations**: Use batch processing for bulk inserts/updates
- **Connection Pooling**: Use HikariCP, Apache DBCP for production

Common JDBC Exceptions

Exception	Cause
SQLException	General database access error
SQLTimeoutException	Query execution timeout
SQLSyntaxErrorException	SQL syntax error
SQLIntegrityConstraintViolationException	Constraint violation (duplicate key, foreign key)
SQLDataException	Invalid data type or format
SQLNonTransientConnectionException	Connection failure that won't resolve automatically

Table 4: Common JDBC Exceptions

Key Takeaways

- **JDBC Architecture**: Application → DriverManager → Driver → Database
- **Connection Steps**: Load driver → Get connection → Create statement → Execute query → Process results → Close resources
- **Statement Types**: Statement (static SQL), PreparedStatement (parameterized), CallableStatement (stored procedures)
- **ResultSet Types**: Forward-only, scrollable, updatable

- **Transaction Management**: ACID properties with commit/rollback
- **Performance**: Use PreparedStatement, batch processing, connection pooling
- **Security**: Always use PreparedStatement to prevent SQL injection
- **Error Handling**: Handle SQLException with proper rollback on failures

MS Access Database Connectivity with JDBC

Setting up MS Access Database

MS Access databases (.accdb or .mdb files) can be accessed using the UCanAccess JDBC driver, which is the recommended approach for modern Java applications (the JDBC-ODBC bridge is deprecated).

Step 1: Download UCanAccess Driver

Download the UCanAccess JAR files from the official repository or Maven:

- `ucanaccess-5.0.1.jar`
- `commons-lang3-3.8.1.jar`
- `commons-logging-1.2.jar`
- `hsqldb-2.5.0.jar`
- `jackcess-3.0.1.jar`

Alternatively, add Maven dependency:

```
<dependency>
    <groupId>net.sf.ucanaccess</groupId>
    <artifactId>ucanaccess</artifactId>
    <version>5.0.1</version>
</dependency>
```

Listing 2: Maven Dependency for UCanAccess

Step 2: Database File Setup

1. Open Microsoft Access
2. Create a new database or use existing one
3. Save with name (e.g., `EmployeeDB.accdb`)
4. Note the absolute path of the database file

Step 3: Connection String Format

The connection string format for MS Access using UCanAccess:

`jdbc:ucanaccess://path/to/database.accdb`

Connection Examples

Example 1: Basic Connection to MS Access

```java
import java.sql.*;

public class AccessConnectionDemo {
    public static void main(String[] args) {
        // Path to MS Access database file
        String databasePath = "C:/MyDatabase/EmployeeDB.accdb";
        String url = "jdbc:ucanaccess://" + databasePath;

        try {
            // Load driver (optional in JDBC 4.0+)
            Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");

            // Establish connection
            Connection conn = DriverManager.getConnection(url);
            System.out.println (" Connected to MS Access database!");

            // Test query
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT 1 FROM MSysObjects");

            if (rs.next()) {
                System.out.println (" Database query successful!");
            }

            // Close resources
            rs.close();
            stmt.close();
            conn.close();

        } catch (ClassNotFoundException e) {
            System.err.println("UCanAccess driver not found: " + e.getMessage());
        } catch (SQLException e) {
            System.err.println("Database error: " + e.getMessage());
        }
    }
}
```

Listing 3: MS Access Basic Connection

Example 2: Connection with Additional Parameters

```java
import java.sql.*;

public class AccessConnectionWithParams {
    public static void main(String[] args) {
        // Connection parameters
        String databasePath = "C:/MyDatabase/EmployeeDB.accdb";
        String username = "";  // MS Access typically doesn't use username
        String password = "";  // Set if database has password

        // URL with additional parameters
        String url = "jdbc:ucanaccess://" + databasePath +
                     ";memory=false" +           // Don't load entire DB in memory
                     ";showSchema=true" +        // Show schema information
                     ";jackcessOpener=crypt" +   // For encrypted databases
                     ";singleConnection=true";   // Single connection mode

        try (Connection conn = DriverManager.getConnection(url, username, password)) {
            System.out.println (" Connected with parameters!");

            // Database operations here

        } catch (SQLException e) {
            System.err.println("Connection failed: " + e.getMessage());
        }
    }
}
```

Listing 4: MS Access Connection with Parameters

Complete MS Access Database Example

```
import java.sql.*;
import java.util.*;

public class MsAccessDemo {

    // Database connection details
    private static final String DB_PATH = "C:/MyDatabase/CompanyDB.accdb";
    private static final String URL = "jdbc:ucanaccess://" + DB_PATH;

    public static void main(String[] args) {

        System.out.println("=== MS Access Database Demo ===\n");

        // Test connection and perform operations
        try (Connection conn = DriverManager.getConnection(URL)) {
            System.out.println (" Connected to: " + DB_PATH);

            // 1. Create table
            createEmployeeTable(conn);

            // 2. Insert records
            insertEmployeeRecords(conn);

            // 3. Query all employees
            queryAllEmployees(conn);

            // 4. Query with condition
            queryEmployeesByDepartment(conn, "IT");

            // 5. Update records
            updateEmployeeSalary(conn, "IT", 10.0);

            // 6. Delete records
            deleteEmployee(conn, "John Doe");

            // 7. Show table structure
            showTableStructure(conn, "Employees");

        } catch (SQLException e) {
            System.err.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private static void createEmployeeTable(Connection conn) throws SQLException {
        String sql = "CREATE TABLE IF NOT EXISTS Employees (" +
                     "EmployeeID AUTOINCREMENT PRIMARY KEY, " +
                     "FirstName VARCHAR(50) NOT NULL, " +
                     "LastName VARCHAR(50) NOT NULL, " +
                     "Department VARCHAR(50), " +
                     "Salary CURRENCY, " +
                     "HireDate DATE, " +
                     "IsActive BIT)";

        try (Statement stmt = conn.createStatement()) {
            stmt.execute(sql);
            System.out.println (" Table 'Employees' created/verified");
        } catch (SQLException e) {
            // Table might already exist
            System.out.println("Note: " + e.getMessage());
        }
    }

    private static void insertEmployeeRecords(Connection conn) throws SQLException {
        String sql = "INSERT INTO Employees (FirstName, LastName, Department, Salary, HireDate
    , IsActive) " +
                     "VALUES (?, ?, ?, ?, ?, ?)";

        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            // Record 1
            pstmt.setString(1, "John");
```

```
            pstmt.setString(2, "Smith");
            pstmt.setString(3, "IT");
            pstmt.setDouble(4, 75000.00);
            pstmt.setDate(5, Date.valueOf("2023-01-15"));
            pstmt.setBoolean(6, true);
            pstmt.executeUpdate();

            // Record 2
            pstmt.setString(1, "Jane");
            pstmt.setString(2, "Doe");
            pstmt.setString(3, "HR");
            pstmt.setDouble(4, 65000.00);
            pstmt.setDate(5, Date.valueOf("2023-02-20"));
            pstmt.setBoolean(6, true);
            pstmt.executeUpdate();

            // Record 3
            pstmt.setString(1, "Bob");
            pstmt.setString(2, "Johnson");
            pstmt.setString(3, "IT");
            pstmt.setDouble(4, 82000.00);
            pstmt.setDate(5, Date.valueOf("2023-03-10"));
            pstmt.setBoolean(6, true);
            pstmt.executeUpdate();

            System.out.println(" Inserted 3 employee records");
        }
    }

    private static void queryAllEmployees(Connection conn) throws SQLException {
        String sql = "SELECT * FROM Employees ORDER BY EmployeeID";

        try (Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            System.out.println("\n--- All Employees ---");
            System.out.println("ID\tFirst Name\tLast Name\tDepartment\tSalary\t\tHire Date");
            System.out.println("---\t---------\t--------\t----------\t------\t\t--------");

            while (rs.next()) {
                System.out.printf("%d\t%s\t\t%s\t\t%s\t\t$%.2f\t%s%n",
                    rs.getInt("EmployeeID"),
                    rs.getString("FirstName"),
                    rs.getString("LastName"),
                    rs.getString("Department"),
                    rs.getDouble("Salary"),
                    rs.getDate("HireDate"));
            }
        }
    }

    private static void queryEmployeesByDepartment(Connection conn, String department)
            throws SQLException {
        String sql = "SELECT * FROM Employees WHERE Department = ?";

        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, department);

            try (ResultSet rs = pstmt.executeQuery()) {
                System.out.println("\n--- " + department + " Department Employees ---");
                while (rs.next()) {
                    System.out.printf("  %s %s - Salary: $%.2f%n",
                        rs.getString("FirstName"),
                        rs.getString("LastName"),
                        rs.getDouble("Salary"));
                }
            }
        }
    }

    private static void updateEmployeeSalary(Connection conn, String department,
                                             double percentage) throws SQLException {
        String sql = "UPDATE Employees SET Salary = Salary * (1 + ?/100) " +
```

```
                    "WHERE Department = ?";

        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setDouble(1, percentage);
            pstmt.setString(2, department);
            int rows = pstmt.executeUpdate();
            System.out.printf("\ n Updated %d employee(s) in %s department (+%.1f%% salary)%n
    ",
                rows, department, percentage);
        }
    }

    private static void deleteEmployee(Connection conn, String name) throws SQLException {
        // Delete by first name (simplified)
        String sql = "DELETE FROM Employees WHERE FirstName = ?";

        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, name);
            int rows = pstmt.executeUpdate();
            System.out.printf("\ n Deleted %d employee(s) named '%s'%n", rows, name);
        }
    }

    private static void showTableStructure(Connection conn, String tableName)
            throws SQLException {
        DatabaseMetaData metaData = conn.getMetaData();

        System.out.println("\n--- Table Structure: " + tableName + " ---");
        ResultSet columns = metaData.getColumns(null, null, tableName, null);

        System.out.println("Column Name\t\tType\t\tSize\tNullable");
        System.out.println("-----------\t\t----\t\t----\t--------");

        while (columns.next()) {
            String colName = columns.getString("COLUMN_NAME");
            String colType = columns.getString("TYPE_NAME");
            int colSize = columns.getInt("COLUMN_SIZE");
            int nullable = columns.getInt("NULLABLE");

            System.out.printf("%-20s\t%-12s\t%d\t%s%n",
                colName,
                colType,
                colSize,
                (nullable == DatabaseMetaData.columnNullable) ? "YES" : "NO");
        }
        columns.close();
    }
}
```

Listing 5: Complete MS Access Database Operations

UCanAccess Connection Parameters

Parameter	Values	Description
`memory`	true/false	Load entire database in memory (faster but uses more RAM)
`showSchema`	true/false	Display schema information in metadata
`singleConnection`	true/false	Use single connection mode
`jackcessOpener`	crypt/none	Handle encrypted databases
`readOnly`	true/false	Open database in read-only mode
`columnOrder`	DISPLAY/NATIVE	Column ordering in result sets

Table 5: UCanAccess Connection Parameters

Common Issues and Solutions

Issue	Cause	Solution
Driver not found	UCanAccess JARs missing	Add all required JAR files to classpath
File locked	Database open in Access	Close Access before connecting
Path not found	Incorrect file path	Use absolute path with forward slashes or double backslashes
Access denied	File permissions	Ensure Java has read/write permissions
Unsupported format	Old .mdb file	Convert to .accdb or use older driver version
Password protected	Database encrypted	Add password parameter or use jackcessOpener

Table 6: Common MS Access Connectivity Issues

Alternative: Setting up ODBC Data Source Name (DSN)

Legacy Approach - JDBC-ODBC Bridge (Deprecated)

Note: The JDBC-ODBC bridge has been removed in Java 8 and is not recommended. This section is for reference only.

Step 1: Create ODBC Data Source (Windows)

1. Open **ODBC Data Source Administrator**:
   - Windows 10/11: Search "ODBC Data Sources"
   - Control Panel → Administrative Tools → ODBC Data Sources
2. Choose **System DSN** tab
3. Click **Add** → Select **Microsoft Access Driver (*.mdb, *.accdb)**
4. Enter **Data Source Name**: e.g., `MyAccessDB`
5. Click **Select** and choose your database file
6. Click **OK** to save

Step 2: Connection String with DSN

```
// JDBC-ODBC Bridge connection (deprecated)
String url = "jdbc:odbc:MyAccessDB";
Connection conn = DriverManager.getConnection(url);
```

Listing 6: Connection using ODBC DSN

Project Setup Instructions

Using Command Line

```
# Compile with UCanAccess JARs
javac -cp "ucanaccess-5.0.1.jar;commons-lang3-3.8.1.jar;commons-logging-1.2.jar;hsqldb-2.5.0.jar;jackcess-3.0.1.jar" MsAccessDemo.java

# Run the program
java -cp ".;ucanaccess-5.0.1.jar;commons-lang3-3.8.1.jar;commons-logging-1.2.jar;hsqldb-2.5.0.jar;jackcess-3.0.1.jar" MsAccessDemo
```

Using Eclipse/IntelliJ IDEA

1. Create a new Java project
2. Right-click project → Build Path → Configure Build Path
3. Click **Add External JARs**
4. Select all UCanAccess JAR files
5. Click **Apply and Close**

Using Maven (pom.xml)

```
<dependencies>
    <dependency>
        <groupId>net.sf.ucanaccess</groupId>
        <artifactId>ucanaccess</artifactId>
        <version>5.0.1</version>
    </dependency>
</dependencies>
```

Verification Checklist

Before running your JDBC program with MS Access, verify:

- Database file exists at specified path
- UCanAccess JARs are in classpath
- Database file is not open in Microsoft Access
- File permissions allow read/write access
- For encrypted databases, password is correct
- Database format is compatible (.accdb or .mdb)
- All required tables exist or creation permissions are granted

Testing Connectivity

Use this simple test program to verify your MS Access connectivity:

```
import java.sql.*;

public class AccessConnectionTest {
    public static void main(String[] args) {
        // Update this path to your database location
        String dbPath = "C:/MyDatabase/TestDB.accdb";
        String url = "jdbc:ucanaccess://" + dbPath;

        System.out.println("Testing connection to: " + dbPath);

        try {
            // Optional: Load driver explicitly
            Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");

            // Attempt connection
            try (Connection conn = DriverManager.getConnection(url)) {
                System.out.println (" CONNECTION SUCCESSFUL!");

                // Test database access
                try (Statement stmt = conn.createStatement()) {
                    ResultSet rs = stmt.executeQuery("SELECT 1");
                    if (rs.next()) {
                        System.out.println (" Database query successful!");
                    }
                    rs.close();
                }
```

```
            }

        } catch (ClassNotFoundException e) {
            System.err.println(" UCanAccess driver not found!");
            System.err.println("Make sure all JAR files are in classpath");
        } catch (SQLException e) {
            System.err.println(" Connection failed!");
            System.err.println("Error: " + e.getMessage());
            System.err.println("SQL State: " + e.getSQLState());
        }
    }
}
```

Listing 7: MS Access Connectivity Test